My file

SCIENCE AND ENGINEERING RESEARCH COUNCIL
RUTHERFORD APPLETON LABORATORY

COMPUTING DIVISION

DISTRIBUTED INTERACTIVE COMPUTING NOTE 799

PERQ UNIX Implementation Note 58
Screen Editor Proposal

Issued by
James Collis

11 February 83

DISTRIBUTION:

R W Witty
K Robinson
C Prosser
A S Williams
L O Ford
T Watson
E V C Fielding
J C Malone
P J Smith
A J Kinroy
J M Loveluck
C P Wadsworth
J R Collis
M Ritchie ICL Dalkeith
P Palmer ICL Dalkeith
RL Support/PERQ/Unix Implementation Notes File

1. Introduction

This document describes the functionality and user interface of a screen editor to be implemented on an ICL PERQ. The editor is a means of examining and altering files in the UNIX environment, making use of the PERQ's high quality display, tablet and mouse. Primarily it is targetted for use by people involved in research for S.E.R.C.

2. Summary

The major aspects of the proposed editor :

A small number of menu orientated commands operating on selected text.
The editor will be modeless.
Visual feedback on everything the user does.
Multiple file editing with easy transfer of data.
Adheres to the philosophy of the UNIX Operating System.
Powerful search and replace capabilities.

If you have any suggestions for improving this editor please contact me on extension 6488 by Friday 25 March.

3. Design Issues

The single most important consideration in the design of an editor is the user. It is the user who will work with the editor, and that is who it should be designed for. The user has many requirements of a screen editor, some of which conflict, and the most difficult part of the design is not so much what to provide, but how to provide the right balance.

User Requirements :-

1) Easy to Learn
2) Easy to Use
3) Safe
4) Fast
5) Powerful

There are several features that make an editor easy to learn. Firstly it should be similar to something the user already knows. There should only be a small number of commands, preferably with meaningful names, or at least mnemonics. Some kind of 'help' facility readily available and everything should be as simple as possible.

Ease of use follows mainly from how easy it was to learn, but now brevity of commands becomes an advantage if they are typed,(though graphical selection may be even better). The display is important, what information is shown on it, and how well it is presented has a strong influence on how pleasant and easy the editor is to use.

Safety for the user means that it is possible to recover from mistakes, either by themselves or the machine, which frees the user from the burden of making sure that each command does exactly what is intended, leaving him free to experiment.

The user doesn't want to sit and twiddle his thumbs while waiting for the editor to finish doing something.

Power comes from the functionality of the editor.

a) Travel, quickly and easily to any part of the file, making it easy to browse through or study closely any particular section of the file.

b) Easy insertion of text anywhere in the file.

c) Simple deletion of any amount of information from a single character to the whole file.

d) To search and replace, forwards or backwards, interactively or not, with the use of wild cards to provide powerful pattern matching.

e) To view and edit more than one file at a time, with easy transfer of data between different files.

4. History

Numerous editors with widely differing styles and philosophies were examined, this section does not attempt to compare or discuss all the features available, but covers the major deficiencies and faults of previous editor designs. For further reference on other editors and editing :

A C M Computing Surveys
Vol 14 number 3 September 1982.

In the past editor interfaces out of necessity, have typically been a mass of shift and control characters typed at the keyboard (simply because this was the only input device). Then with the keyboard characters as commands, editors tended to use different modes to allow for inserting. Also the commands themselves functioned using line ranges as operands, which had to be specified by the user. These characteristics cause difficulties in the user interface:-

Many editors have two modes insert and delete, though others have more (eg. NUROS has 5 & SOS has 7, used for grouping commands). It is confusing to the user, to only have certain commands available in certain modes, or to have to type differently in different modes to execute the same command. It also means the user must remember which is the current mode, and displaying the mode only helps!

For the user to specify the operand of a range in terms of line numbers, patterns and marks involves a lot of work. First you recognise what you want to operate on. Then this has to be converted into some form recognisable by the editor (as above), and also be syntactically correct.

The new hardware available (specifically the tablet), opens up a completely different scope. In an attempt to realise this potential, we selected the best and most popular editor available that uses the mouse, and tried change what we didn't like.

5. Pepper

Although very popular, Pepper only solves the second of the above problems, selecting the operand by pointing at it with the tablet and mouse. A method believed to be much closer to the human approach, and borne out by users opinions. Unfortunately Pepper still has two modes and 75 distinct commands, (the number of commands in an editor have a major effect on the lead in time for learning). Only 9 commands are executed from the puck, the others are combinations of single and double, shifted and control characters providing terrible mnemonics. They also require both hands on the keyboard, which destroys the natural feel of the mouse, and as a result the user interface is spoilt, (see Appendix A).

Some other smaller points about Pepper:

The commands issued from the mouse to traverse the file are very good.

Relative tracking of the mouse is used, which allows operations to take place at the cursor position.

There is no simple way of accessing a particular line number, and while this is not such a natural thing to do, unfortunately it is required for tracing Pascal and C compiler diagnostic messages, which specifically refer to line numbers.

All the keyboard commands to move and a variety of others are redundant, and can be done more easily with the mouse.

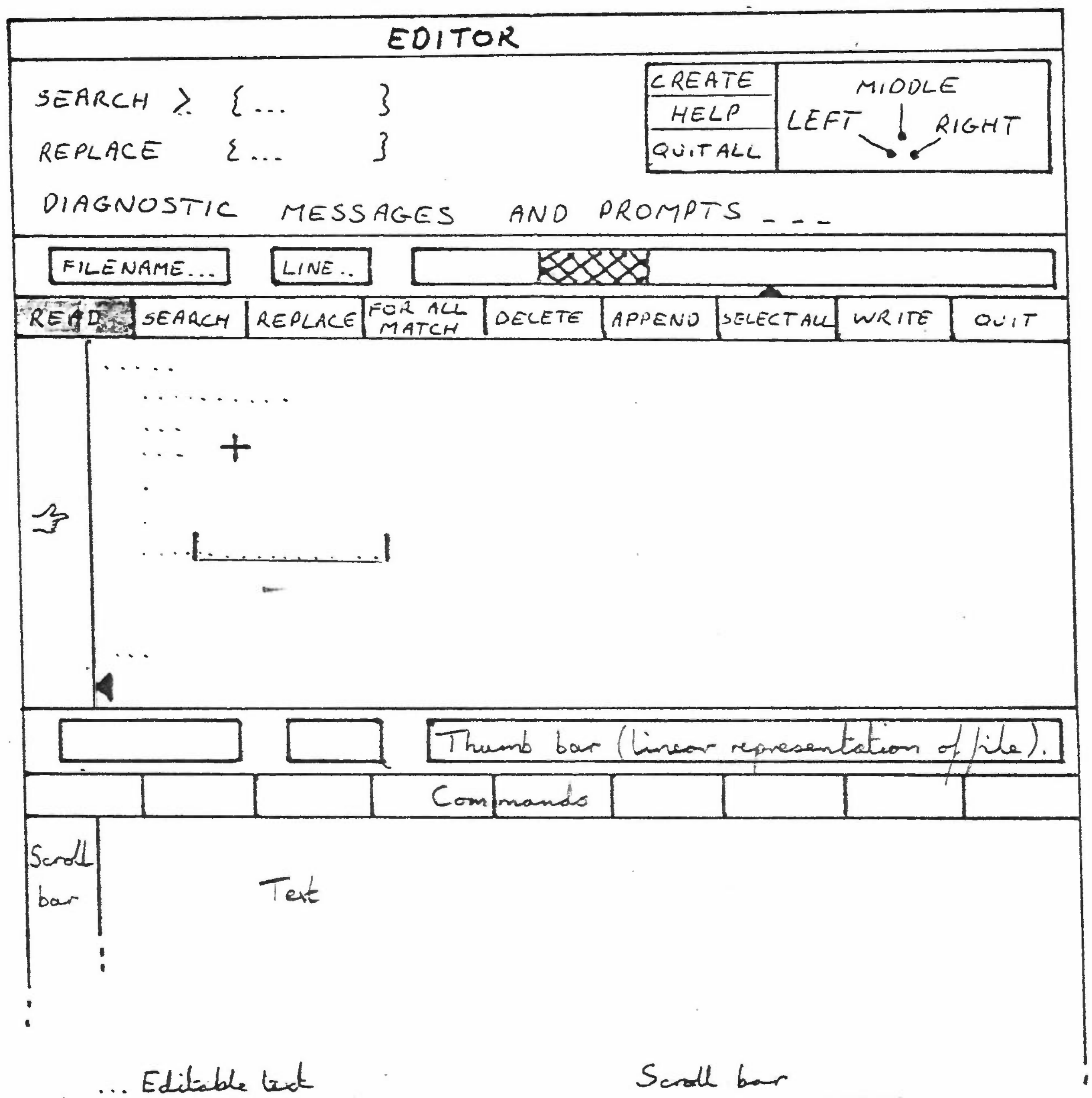
The search and replace functions do not have any pattern matching or meta character capabilities.

6. The Design Phase

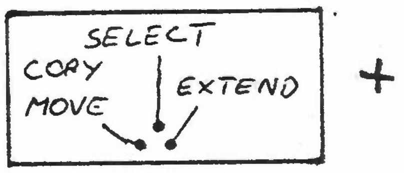
Objectives :-

1) To remove all commands from the keyboard, and thus
2) Perfect the modeless environment where all the keys are consistent.
3) To provide a still smaller set of commands, chosen to be the most basic operations, which can be combined to satisfy all editing requirements.
4) To allow for the possible introduction of the three button puck.

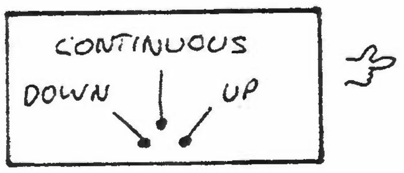
Removing commands from the keyboard requires them to be graphically selectable. (ie. point with mouse and press button.) This adds to the importance of reducing the number of commands that need pointing at to avoid cluttering the screen, (see display).



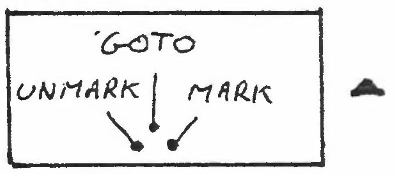
... Editable text



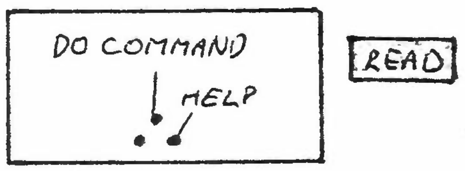
Scroll bar



Thumb bar



Command



End of file = ◀

Selection = | or |____|

Before the commands were decided, each one was carefully scrutinised, and had to justify itself functionally before being included in the final set. For example, the argument for binding move and copy to the mouse:-

With absolute tracking move and copy (if graphically selectable), need a source and destination. If the left button in the text area is set to fix a target for the operand, then this needs to be displayed differently to the cursor and the selection. To maintain consistency with the search and replace commands, the string matched by the search should be the target, (not the selection). So now the target is a string and not just a place, which presents greater problems of different echoing, user complexity and raises the question of ordering between the strings.

To avoid these nasties, the idea of move and copy was swapped for cut, copy and paste. Where:

cut	: cuts the selection to a buffer.
copy	: copies it there.
paste	: sticks the buffer back over the selection.

This brings in arguments of cut buffers versus multi- windows, and it requires extra keystrokes, (due to text having an indirect stop off in buffer), for just the type of operation that should be ideal using the mouse, (ie. pick it up and put it down).

The next solution is to use the left button either as a pop-up menu (offering move and copy), or the actual copy command, both using the cursor position at time of button press as the destination. With absolute tracking, the menu scheme would work, but covers the exact area of interest when appearing. So the left button is set to copy, and a double press within a preset time limit is a move.

7. Command Summary

This section only gives a brief definition of what each command does, and while it describes what happens in some exceptional circumstances, it is not intended to cover everything, though the vast majority of possibilities have been considered.

There is always a selection, at its smallest it is a position between two characters, at most the whole window is selected, (not just the visible window).

Insertion of typed text occurs to the left of the current selection, thus typed strings are inserted in the correct order. Unfortunately this is the only sensible way to type with absolute tracking, (with relative tracking text could be inserted at the cursor).

Command	Function
editor	Invokes the editor, if given an argument it reads that file into the first window, otherwise it presents a blank page, in both cases the position before the first character in the file is selected.

create	Creates a new window where sufficient space is available.
quitall	Quits the editor, flashes a warning on any window not written to a file.
help	Writes a message on how to get help on individual commands. Help on the editor in general is considered to be a problem global to the UNIX operating system.
read	Reads the file in filename as if it were typed at the current selection, and selects the position before the first character of the read in file.
select	Selects the nearest position to the cursor between two characters.
extend	Extends the selection to the nearest position to the cursor between two characters.
copy	Duplicates the selected text at the nearest position to the cursor between two characters.
move	Moves the selected text to the cursor position as above.
down	Scrolls the top line in the window to the current line.
up	Scrolls the current line in the window to the top line.
continuous	Scrolls up or down continuously depending on movement after initial press until button released.
goto	Displays in the window, the approximate region of the file pointed at.
mark	Puts a mark on the thumb bar.
unmark	Removes the mark pointed at on the thumb bar. (N.B. This type of marking was decided upon with an analogy to book marks. For example, imagine several pieces of blank paper sticking out of a book - they only mark the context not an exact position.)
selectall	Selects the whole window.
delete	Deletes the selected text and places it in the bit bucket, (except when it is deleted from the bit bucket).
append	Appends the selected text to file in filename.
search	Selects the first text from the current selection, in the direction specified by < or >, that is matched by the contents of the search buffer.
replace	Replaces the selection with the contents of the replace buffer. (N.B. For search and replace operations, meta characters with special meanings may be used. These characters are the same as in System 3 regex(3) apart from the fact that they are now control characters.)
forallmatch	Selects the first match of the search buffer, then if any other command is done immediately after, the same command is done for every match. To abort after the first match just select something else.
quit	Quits this window, if it has not been written, then when the button is depressed, a warning message flashes, if the button is released before the cursor is moved away, the window quits, otherwise the quit was aborted.
docommand	Executes the highlighted command as defined.
help	Informs the user what the highlighted command will do if executed, and how to use it.

8. Clarification of some minor points.

Selecting beyond the end of a line pads out with spaces.

Scrolling beyond end of file stops when EOF marker is at top of window.

Selection beyond EOF pads out with new lines, then if that position is also beyond end of line, that line is padded as before.

Moving the selected text into itself does nothing.

Copying the selected text into itself includes shifted text in the selection.

Delete does not remove the selection, merely changes it to the position between characters previously at each end of the selection.

Tabstops are fixed every 8 characters, tabs are not expanded to spaces but appear as such.

Auto indent on LineFeed, using the tabs and spaces from the previous line.

tty settings for interupt, quit, DEL, OOPS, etc. are maintained.

Control characters are echoed as the inverse of normal characters, (ie. white on black).

Safety from user mistakes is provided by a file called bit.bucket, which is created when the editor is invoked, and removed when the editor is exited. The bit bucket contains copies of all text that is deleted. This file is also editable, but anything deleted from it is not saved. There is only one bit.bucket for the whole editing session.

In general, any command may be aborted by moving the mouse away from where it was pressed, before releasing.

If the mouse or tablet fails to function, no backup is provided.

The search, replace, line number and file name buffers are editable as text. Editing the line number places the desired line in the center of the page.

Changing the window size will be another mouse operation, with the cursor appearing differently between the line number and thumb bar, after one press the window will be altered by the vertical distance then moved before the next press.

There will be no escape to the shell as this is unnecessary with the window manager.

The ability to pipe certain lines to another process is being considered, but left as a possible future enhancement.

Appendix A.

Pepper Command Summary:

Key	Command
f	forward character
F	forward word
b	backward character
B	backward word
a	begin line
e	end line
p	up line
n	down line
V	up page
v	down page
,	top of window
.	bottom of window
x,	begin file
x.	end file
=	position to selection
@	set mark
xx	find mark
yellow	select characters
	scroll current line to top
white	select words
	continuous scrolling
	thumb to page
green	select lines
	continuous scrolling
blue	alter selection
	scroll top line to current line
*	select entire file
i, TAB	insert tab
CR	new line
LF	new line and indent
q	quote next character
<space>	insert a space
t	twiddle characters
o	open space
O	open space and indent
y	yank kill buffer
Y	pop kill buffer
'	insert selection
d	delete character
D	delete word { ctrl-D also gives stack dump }
h, BS	delete previous character
H	delete previous word
k	delete to end of line
OOPS	delete to start of line
K	delete back to indentation
"	delete selection

x0	enter search string
x1	enter replace string
x2	search down
x3	search up
x4	replace down
x5	replace up
s	enter string and search down
r	enter string and search up
R	enter strings and replace down
xR	enter strings and replace up
x6	toggle insert mode
x_	enter/exit scroll region
x^	enter/exit thumb region
xf	exit editor
c, C	immediate exit
HELP, ?	get this help file
xs	write current file
xS	write backup
xw	write named file
l	refresh screen
xv	make window
xd	kill window
xZ	enlarge window
xz	shrink window
xr	replace window
1..9	goto window 1..9
X	change parameter
u	enter repeat count
INS	repeat last command
DEL	abort command